Pet clinic mandatory assignment report

Krisztian Szabo

November 12, 2019

`https://github.com/SzaboKrisztian/PetClinic`

Contents

1 Introduction

In this project, I employed the following technologies:

- **Java**: A high level object-oriented programming language that runs on the Java Virtual Machine.
- **Spring**: an open source Model-View-Controller based application framework for Java, that is meant to facilitate the development of web applications. Spring boot is an opinionated approach that is mean to help the developer get going fast, by providing sensible defaults and including what are considered to be the best and most used third party libraries.

  The Model-View-Controller (henceforth referred to as MVC) is a software design pattern, or as some call it, a high-level software architecture pattern, that is widely used in developing applications that feature a user interface. Its purpose is to decouple the business logic (the models) from the user interface (the views) and provide a means of communication between these two elements (the controllers). There are variations as to how the MVC pattern can be implemented, but in the more classic approach, the controller is the entry point to the program. This instantiates the required models and views, to which it passes a reference of itself, to facilitate communication from them back to the controller. The user only interacts with the views, which have the role of displaying themselves, and of accepting user input. This input is processed by the controller, which decides what action to take, like, for example, changing some values in the model.

  The MVC pattern achieves great decoupling between the different components, and allows for the models and views to be reused.
- **Thymeleaf**: A powerful and very flexible templating engine for Spring.
- **Tomcat**: A pure Java HTTP web server which can run Java code.
- **Maven**: A build automation tool primarily used for Java projects. This allows for easy management of a projects dependencies.
- **Git**: The most popular version control tool to keep track of changes to a code base.

2 Implementing the changes

I already had my development environment set up, with JDK, Maven, and git correctly installed and added to the environment variables. Cloning the repository to a local folder was also trivial.

The first real step in tackling this assignment was to identify where to make changes within the codebase. By default, the Pet Clinic web application uses **hsqldb** as data storage. I found out that it works as an in-memory database, which creates its structure and populates it with data at runtime. So the place to modify the table structure is in the **src/main/resources/db/hsqldb/schema.sql** file and adding data to populate it in the **src/main/resources/db/hsqldb/data.sql** file.

The next thing to be modified was to add a new menu entry on the navigation bar for the future view that needed to be created. I found the navigation bar as a thymeleaf fragment in the **src/main/resources/templates/fragments/layout.html** file. I added an additional entry and pointed it to "/drugs" mapping, which for the moment does not yet exist.

Next task was to define the **Drug** class, which in our case should extend the existing **NamedEntity** class. It has to define the **type**, **batchNumber**, and **expiryDate** attributes, and bind them to their respective SQL columns. The **id** and **name** attributes are taken care of in the **BaseEntity** and **NamedEntity** superclasses, respectively.

Then, I defined the **DrugRepository** interface, which defines some basic methods for retrieving and saving **Drug** entities from and to the database. Having done this, I moved to writing the **DrugController** class. In it, I added an instance of **DrugRepository** and defined the GET mapping for "/drugs". All this does is it inserts a list of all the available **Drug** entities into the model, and returns the **src/main/resources/templates/drugs/drugs.html** view. Next I went ahead and actually created the view, which consists of a simple html table which is populated by thymeleaf by iterating over the list of **Drug** entities that was sent from the controller through the model.

After getting this to work, I realized that it would make logical sense to define a many-to-many relationship between the newly created drugs and the animal types, as one drug can most likely be used for several animals. To achieve this, I added a new table to the **schema.sql** file, that only serves the purpose of mapping this relationship, by the means of having only two columns, which are each a foreign key pointing to the **id** columns of the **drugs** and **types** tables, respectively. I then went ahead and modified the **Drugs** POJO to hold, instead of a single **PetType** attribute, a **Set<PetType>**. The last change needed to be made was in the view, for each drug displayed in the table, to iterate over all of its associated **PetType** objects and show them in a nicely formatted string.

Inspired by the way thymeleaf templates are used throughout the project to generate and populate web forms, I then went ahead and wrote **src/-main/resources/templates/drugs/addOrUpdateDrug.html** in a similar manner, to serve as the view for both adding and updating **Drug** entries. For this to work, I next had to define the four new mappings in the **DrugController** class, a *GET* and a *POST* mapping for both the add and update functionality, and I added the relevant links in the main **drugs.html** view, namely an *"Add new drug"* button, and an *"Edit"* and a *"Delete"* link for each entry in the table. I also added a **DrugValidator** class to handle data validation in these two new views.

3 Final thoughts

The process of making changes to this sample Spring project was quite enlightening, and I feel that I learned a lot from it. I got to see for the first time a code base that uses Spring's capabilities to the maximum, and I now realize that last semester's project could have been done in a much more efficient manner (though, to be fair, there were some limitations imposed on us by our teachers, for specific purposes). The most difficult part, so to speak, was understanding the code base. This took up the greatest part of my time spent on this project. Otherwise, after having understood what the original authors have built, implementing new functionality in the same style as theirs was fairly straight-forward.

I did hit a snag at one point, which took me a good few hours to debug. It had to do with getting Spring / Thymeleaf to correctly bind the multi-select box in the **addOrUpdateDrug.html** view to the **Drug** class' **petType** attribute, which is a collection of the type **Set<PetType>**. I didn't find any relevant documentation on this behavior, but apparently, when binding a web form to an object, Spring expects the POJO to have plain getters and setters for the attributes that are to be bound. I had originally written a slightly more complicated setter for this collection attribute in the **Drug** class, inspired by the authors' **Owner** class, and this lead Spring not being able to insert the data submitted in the web form into the POJO's respective attribute.

4 Code listings

'schema.sql'

```sql
CREATE TABLE drugs (
  id              INTEGER IDENTITY PRIMARY KEY,
  name            VARCHAR(80),
  batch_number    VARCHAR(30),
  expiry_date     DATE
);
CREATE INDEX drugs_name ON drugs (name);

CREATE TABLE drugs_for_animal (
  drug_id         INTEGER NOT NULL,
  animal_type_id INTEGER NOT NULL
);
ALTER TABLE drugs_for_animal ADD CONSTRAINT
   fk_drugs_for_animal_types FOREIGN KEY (animal_type_id)
   REFERENCES types (id);
ALTER TABLE drugs_for_animal ADD CONSTRAINT
   fk_drugs_for_animal_drug FOREIGN KEY (drug_id) REFERENCES
   drugs (id);
```

'Drug.java'

```java
@Entity
@Table(name = "drugs")
public class Drug extends NamedEntity {

  @ManyToMany(fetch = FetchType.EAGER)
  @JoinTable(name = "drugs_for_animal", joinColumns =
     @JoinColumn(name = "drug_id"), inverseJoinColumns =
     @JoinColumn(name = "animal_type_id"))
  public Set<PetType> petTypes;

  @Column(name = "batch_number")
  private String batchNumber;

  @Column(name = "expiry_date")
  @DateTimeFormat(pattern = "yyyy-MM-dd")
  private LocalDate expiryDate;

  public void setPetTypes(Set<PetType> petTypes) {
    this.petTypes = petTypes;
  }

  public Set<PetType> getPetTypes() {
    return this.petTypes;
  }

  public int getNrOfPetTypes() {
    return this.petTypes.size();
  }

  public String getBatchNumber() {
    return batchNumber;
  }

  public void setBatchNumber(String batchNumber) {
    this.batchNumber = batchNumber;
  }

  public LocalDate getExpiryDate() {
    return expiryDate;
  }

  public void setExpiryDate(LocalDate expiryDate) {
    this.expiryDate = expiryDate;
  }
}
```

'DrugController.java'

```java
@Controller
public class DrugController {

  private final String ADD_OR_UPDATE_DRUG_VIEW = "drugs/
     addOrUpdateDrug";

  private final DrugRepository drugRepo;
  private final PetRepository pets;

  public DrugController(DrugRepository drugs, PetRepository
     pets) {
    this.drugRepo = drugs;
    this.pets = pets;
  }

  @InitBinder
  public void setAllowedFields(WebDataBinder dataBinder) {
    dataBinder.setDisallowedFields("id");
  }

  @InitBinder("drug")
  public void initDrugBinder(WebDataBinder dataBinder) {
    dataBinder.setValidator(new DrugValidator());
  }

```

```java
@ModelAttribute("allPetTypes")
public Set<PetType> populatePetTypes() {
  return new TreeSet<>(this.pets.findPetTypes());
}

@GetMapping("/drugs")
public String seeDrugsList(Map<String, Object> model) {
  model.put("drugs", drugRepo.findAll());
  return "drugs/drugsList";
}

@GetMapping("/drugs/new")
public String initAddDrugForm(Map<String, Object> model) {
  model.put("drug", new Drug());
  return ADD_OR_UPDATE_DRUG_VIEW;
}

@PostMapping("/drugs/new")
public String processAddDrugForm(@Valid Drug drug,
    BindingResult result, Map<String, Object> model) {
  if (result.hasErrors()) {
    model.put("drug", drug);
    return ADD_OR_UPDATE_DRUG_VIEW;
  } else {
    this.drugRepo.save(drug);
    return "redirect:/drugs";
  }
}

@GetMapping("/drugs/edit/{drug_Id}")
public String initUpdateDrugForm(@PathVariable("drug_Id")
    int drug_Id, Map<String, Object> model) {
  Drug drug = this.drugRepo.findById(drug_Id);
  model.put("drug", drug);
  return ADD_OR_UPDATE_DRUG_VIEW;
}

@PostMapping("/drugs/edit/{drug_Id}")
public String processUpdateDrugForm(@Valid Drug drug,
    @PathVariable("drug_Id") int drug_Id, BindingResult
    result, Map<String, Object> model) {
  if (result.hasErrors()) {
    model.put("drug", drug);
    return ADD_OR_UPDATE_DRUG_VIEW;
  } else {
    drug.setId(drug_Id);
    this.drugRepo.save(drug);
    return "redirect:/drugs";
  }
  }

  @GetMapping("/drugs/delete/{drug_Id}")
  public String deleteDrug(@PathVariable("drug_Id") int
     drugId) {
    this.drugRepo.deleteDrugById(drugId);
    return "redirect:/drugs";
  }
}
```

'drugsList.html'

```
<!DOCTYPE html>

<html xmlns:th="https://www.thymeleaf.org"
      th:replace="~{fragments/layout :: layout (~{::body},'
         drugs')}">

<body>

<h2>Drugs</h2>

<table id="drugs" class="table table-striped">
    <thead>
    <tr>
        <th style="width: 200px;">Name</th>
        <th>Pet type</th>
        <th>Batch number</th>
        <th style="width: 120px">Expiry date</th>
        <th></th>
        <th></th>
    </tr>
    </thead>
    <tbody>
    <tr th:each="drug : ${drugs}">
        <td th:text="${drug.getName()}"></td>
        <td>
            <span th:if="${drug.getNrOfPetTypes()} > 0"
                  th:each="petType, iter : ${drug.getPetTypes
                     ()}"
                  th:text="${petType.getName()} + (!${iter.
                     last} ? ', ' : '')"></span>
            <span th:unless="${drug.getNrOfPetTypes()} > 0">
               none</span>
        </td>
        <td th:text="${drug.getBatchNumber()}"></td>
        <td th:text="${drug.getExpiryDate()}"></td>
        <td>
            <a th:href="'/drugs/edit/' + ${drug.getId()}">
```

```
                Edit</a>
        </td>
        <td>
            <a th:href="'/drugs/delete/' + ${drug.getId()}">
                Delete</a>
        </td>
    </tr>
    </tbody>
</table>

<a href="/drugs/new" class="btn btn-default">Add New Drug</a>

</body>
</html>
```

'addOrUpdateDrug.html'

```
<html xmlns:th="https://www.thymeleaf.org"
     th:replace="~{fragments/layout :: layout (~{::body},'
        drugs')}">

<body>

<h2>
    <th:block th:if="${drug.isNew()}">New </th:block>
    <th:block th:unless="${drug.isNew()}">Edit </th:block>
    Drug
</h2>

<form th:object="${drug}" class="form-horizontal" method="
   post">
     <!-- th:action="${drug.isNew()} ? '/drugs/new' : '/drugs
        /edit/' + ${drug.getId()}" -->

    <div class="form-group has-feedback">
        <input th:replace="~{fragments/inputField :: input ('
           Name', 'name', 'text')}" />

        <input th:replace="~{fragments/inputField :: input ('
           Batch number', 'batchNumber', 'text')}" />

        <input th:replace="~{fragments/inputField :: input ('
           Expiry date', 'expiryDate', 'date')}" />

        <input th:replace="~{fragments/multiSelectField ::
           multiSelect ('Pet types', 'petTypes', ${
           allPetTypes})}" />
    </div>
    <div class="form-group">
        <div class="col-sm-offset-2 col-sm-10">
            <input type="hidden" name="id" th:value="*{id}" /
               >
            <button th:with="text=${drug.isNew()} ? 'Add drug
               ' : 'Update drug'"
                    class="btn btn-default" type="submit" th:
                       text="${text}"></button>
        </div>
    </div>
</form>

</body>
</html>
```